"Express Mail" mailing label number:

EL830058139US

CONFIGURABLE MEDIA-INDEPENDENT SERVER

Anil K. Annadata
Mingtse Chen

CROSS-REFERENCE TO RELATED APPLICATIONS

This application relates to application serial no. _____ (attorney docket M-11404 US), filed on same day herewith, entitled "System and Method for Multi-Channel Communications Queuing" and naming Anil K. Annadata, Wai H. Pak, and Rohit Bedi as inventors, the application being incorporated herein by reference in its entirety.

This application relates to application serial no. _____ (attorney docket M-11405 US), filed on same day herewith, entitled "System and Method for Maintaining Real-Time Agent Information for Multi-Channel Communication Queuing" and naming Anil K. Annadata, Wai H. Pak, and Mingtse Chen as inventors, the application being incorporated herein by reference in its entirety.

This application relates to application serial no. _____ (attorney docket M-11530 US), filed on same day herewith, entitled "Adaptive Communication Application Programming Interface" and naming Mingtse Chen, Anil K. Annadata, and Leon Chan as inventors, the application being incorporated herein by reference in its entirety.

This application relates to application serial no. _____ (attorney docket M-11528 US), filed on same day herewith, entitled "User Interface for Multi-Channel Communication" and naming Mingtse Chen, Anil K. Annadata, and Kuang Huang as inventors, the application being incorporated herein by reference in its entirety.

This application relates to application serial no. _____ (attorney docket M-11529 US), filed on same day herewith, entitled "Multi-Channel Media Independent Server" and naming Mingtse Chen, Anil K. Annadata, and Leon Chan as inventors, the application being incorporated herein by reference in its entirety.

This application relates to application serial no. _____ (attorney docket M-11538 US), filed on same day herewith, entitled "An Extensible Interface for Inter-Module Communication" and naming Wai H. Pak as inventor, the application being incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to communication using multiple communication channels of different media types.

Description of the Related Art

In today's emerging technological and information world, companies are interacting with their customers, potential customers and other contacts through a wide variety of different communication channels. Such communication channels include face-to-face, telephone, fax, email, voicemails, wireless communication, Internet information inquiries via call me now and call me later, Internet collaborative sessions, paging and short messaging services. With all these communication channels, companies are faced with managing each customer interaction while meeting service levels and maximizing customer satisfaction. In addition, companies are faced with optimally staffing and training their workforce to deal with customers through these communication channels whether through their customer support center(s), telebusiness organizations, or their sales, marketing, and service professionals.

Currently, many companies have dedicated email inboxes, fax inboxes, and voicemail boxes defined for specific business areas as well as automated call distributors. Employees called agents are assigned to poll and manage the support requests from customers for each communication channel. Combined with the traditional call queues for inbound telephone calls, each agent is tasked with managing his or her work using all these communication channels while not having any visibility to the queue status and priorities of each customer support request and/or communication channel.

Most communication software is designed to work with a single communication device or type of communication channel. If a company wishes to implement a customer support center where agents can communicate using multiple communication channels of different media types, typically the company must purchase different software products to handle each media type because of the different communication protocols involved. For example, normally an email server is sold separately from software that can receive data via wireless access protocol. Because different products must be purchased, agents must learn to use a different user interface for each media type of the multiple communication channels. Efficiency of an agent typically degrades when he or she must remember different user interfaces for communicating with customers via different media types.

With customer support centers handling very large numbers of customer support requests daily, increasing the efficiency of each agent in responding to each customer request by only seconds can produce enormous cost savings for the customer support center.

Thus, it is desirable to provide a system that includes a universal queue strategy capable of assigning, routing, and queuing work items from multiple channels of communications to an agent having the appropriate skills to respond to the request. The system should enable the agent to view and manage his or her work items for all communication channels. Such a system reduces the response times and increases customer satisfaction, while balancing priorities amongst work items in multiple communication channels.

SUMMARY OF THE INVENTION

The present invention provides a configurable system for communicating via multiple communication channels of different media types. The system includes multiple channel drivers for communicating with a corresponding communication channel of multiple communication channels of diferent media types. The system includes a command table for storing commands associated with a channel driver. The system user interface for specifying an outbound communication channel to perform an outbound command of the plurality of commands. The system includes instructions for determining an outbound channel driver of the plurality of outbound channel drivers to issue the outbound command to the outbound

communication channel. The system includes instructions for sending the outbound command to the outbound channel driver for issuing the outbound command to the outbound communication channel. The system includes a database comprising a command table comprising a plurality of command records, each command record representing a command of the plurality of commands; a channel driver table comprising a plurality of channel driver records, each channel driver record providing a location of a channel driver file name for one of the plurality of channel drivers; and a user interface object table comprising a plurality of user interface object records, each user interface object record corresponding to a command record of the plurality of command records of the command table. The system includes instructions for associating a command of the plurality of commands with a user interface object of the user interface object table, wherein the activating the user interface object issues the associated command.

The foregoing is a summary and thus contains, by necessity, simplifications, generalizations and omissions of detail; consequently, those skilled in the art will appreciate that the summary is illustrative only and is not intended to be in any way limiting. Other aspects, inventive features, and advantages of the present invention, as defined solely by the claims, will become apparent in the non-limiting detailed description set forth below.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be better understood, and its numerous objects, features and advantages made apparent to those skilled in the art by referencing the accompanying drawings.

The use of the same reference symbols in different drawings indicates similar or identical items.

Figs. 1A through 1D are a diagram of one embodiment of a system for enabling and scheduling agents to respond to customer support requests and/or information requests via multiple communication channels of different media types.

Fig. 1E is a diagram of another embodiment of a system for enabling and scheduling agents to respond to customer support requests and/or information requests via multiple communication channels of different media types.

Fig. 1F is a diagram of components included in an implementation of a communication application programming interface.

Fig. 1G is a diagram of components included in another implementation of a communication application programming interface.

Fig. 1H is a diagram of components included in another implementation of a communication application programming interface.

Fig. 1I is a diagram of components included in another implementation of a communication application programming interface.

Fig. 1J is a diagram of components included in another implementation of a communication application programming interface.

Fig. 1K is a diagram of components included in another implementation of a communication application programming interface.

Fig. 2 shows an example of a database schema for the system of Figs. 1A through 1K.

Figs. 2a through 2cc show examples of tables corresponding to table names in Fig. 2.

Fig. 3 is a block diagram showing the processing of commands and events by the system of Figs. 1A through 1K.

Fig. 4 is an example of the operation of components of the system of Figs. 1A through 1K to establish a web collaboration session between a customer and an agent.

Fig. 5 is an example of the operation of components of the system of Figs. 1A through 1K using the universal queuing system component to assign an agent to an incoming telephone call and routing the telephone call to the assigned agent.

Fig. 6 is an example of an embodiment of toolbar 105.

DETAILED DESCRIPTION

The following is intended to provide a detailed description of an example of the invention and should not be taken to be limiting of the invention itself. Rather, any number

of variations may fall within the scope of the invention which is defined in the claims following the description.

Figs. 1A through 1D are a diagram of one embodiment of a client/server system 100 for enabling agents to respond to customer support requests and/or information requests via multiple communication channels of different media types. These media types include, but are not limited to, telephone, email, fax, web collaboration, Internet call me now and call me later, web chat, wireless access protocol, paging, and short messaging services. The term customer is used herein to include individuals and contact persons at businesses that are customers of the company, potential customers and other persons with whom a customer support agent communicates.

Fig. 1A shows that four customers have submitted customer support requests to the client/server system 100 and three agents are responding to customer support requests. The four customers submitted the customer support requests via four communication channels 130, such as communication channels 130A, 130B, 130C, and 130D. In one embodiment, at least two of the four communication channels support different media types.

In accordance with the present invention, client/server system 100 includes a universal queuing (UQ) system 102 capable of assigning, routing, and queuing work items from multiple channels of communication to an agent having the appropriate skills to respond to a customer support request. The term work item refers to a request from a customer that requires a response from an agent assigned by client/server system 100, such as responding to a customer support request in the form of a telephone call, email, fax or other communication of a different media type. A work item can be initiated when an event such as an incoming customer support request arrives or by an agent using a user interface to client/server system 100.

Client/server system 100 also includes a communication server 109 that enables agents to use communication channels of different media types to communicate with customers. Communication server 109 handles events such as the arrival of incoming customer support requests from a channel driver 120 such as one of channel drivers 120A, 120B, and 120C. Each channel driver 120 communicates with a communication channel 130 such as one of communication channels 130A, 130B, 130C and 130D.

Interaction between UQ system 102 and communication server 109 occurs when, for example, communication server 109 receives and routes an incoming customer request as a work item to UQ system 102 for assignment to an agent. UQ system 102 assigns an agent to the work item and identifies an assigned agent to communication server 109 for communication concerning the work item to the assigned agent.

Web browser client 104A includes a web browser program such as Microsoft's Internet Explorer running on a client computer system (not shown). The web browser client 104A communicates with a web server 188. Application server 126 in client/server system 100 performs functions for and sends information to web browser client 104A via web server 188, which provides web pages for web browser client 104A to display. Web server 188 can download program instructions, such as Java applet 116, to the web browser client 104A to provide additional functionality, such as a user interface.

Web browser client 104A is shown including a toolbar 105. One of skill in the art will recognize that other user interfaces providing the functionality of toolbar 105 can be implemented using a variety of different display formats to interface with multiple communication channels of different media types within the scope of the invention. Toolbar 105 is presented as part of a user interface.

In one embodiment, application server 126 of client/server system 100 includes object manager 107, session mode communication server 110, request mode communication server 140, inbound communication receiver 170, UQ system 102, web server 188, web server 146, Enterprise Application Interface (EAI) object manager 190, and workflow process 144. In one embodiment, communication between components in application server 126 is enabled using a suitable inter-process communication protocol in conjunction with transfer control protocol/Internet protocol (TCP/IP) as known in the art.

UQ business service 106 allows communication server 109 to request information from UQ system 102, which returns the information via web server 146, and EAI object manager 190. In one embodiment, both session mode communication server 110 and inbound communication receiver 170 can communicate with UQ system 102. Other embodiments can communicate with a third party queuing system for maintaining work item queues and assigning agents to work items.

Communication server 109 includes at least one of session mode communication server 110, request mode communication server 140, and inbound communication receiver 170. It is important to note that the functionality provided by servers 110, 140, and 170 can be implemented on one server computer system or distributed across two or more server computer systems. Communication server 109 handles all communication between agents and customers via communication channels 130 of one or more media types. Communication server 109 is not media-specific and has no knowledge of communication channels or media.

To communicate with multiple communication channels of different media types, communication server 109 is designed to communicate with a channel driver 120 such as one of channel drivers 120A, 120B, and 120C. A channel driver 120 is written according to Communication Application Program Interface (API) 125. Communication API 125 provides an interface for third party vendors of communication devices and software (e.g., middleware vendors for communication devices) to provide a channel driver 120 so that their products are compatible with application server 126. By implementing a channel driver 120, vendors can take advantage of the customer support center management features and multi-media communication channel capabilities of application server 126.

Communication API 125 is designed to provide flexibility to third party vendors for integrating their products. In the implementation of a channel driver, a vendor defines the commands the vendor's communication channel 130 understands so that communication server 109 can issue commands for the communication channel 130 to perform. Normally these commands are issued when session mode communication server 110 is presenting a user interface to the agent, although inbound communication receiver 170 also can send commands in some circumstances.

In addition, the vendor defines the events that the vendor's communication channel 130 provides regarding activity of a specific communication channel 130. Finally, the vendor provides a channel driver 120 implementation, such as a dynamic link library (.DLL file), for performing each command and generating and providing each event. The channel driver 120 implementation is required by communication API 125 to include code to instantiate a driver object and at least one service object.

By requiring the vendor to provide facilities for the communication server 109 to issue commands to and to receive information from the vendor's communication channel 130, communications API 125 enables communications server 109 to operate independently of the communication channel 130 media type and specific protocols to communicate with the vendor's communication device or software.

Referring to Fig. 2, an example of a database schema 200 that can be used by client/server system 100 (Fig. 1) for storing and communicating channel driver information, agent limitations on media access, commands and events, inbound task management, agent preferences, agent status, media status, communication channel configurations, multiple queue support, and agent management is shown. Database schema 200 includes data structures for configuration base 202, command and event 204, system base 206, response group 208, and email profile access control 210.

Figs. 2a through 2cc show examples of tables corresponding to table names in Fig. 2. Note that Fig. 2 does not indicate all of the relationships between the tables for simplicity, and that many instances of a table may exist for a particular configuration, depending on the number and types of communication channels authorized. Additionally, one skilled in the art will realize that this collection of tables, the parameters included in each table, and the storage space allowed for the parameters, is one example of how the database schema may be configured, and that other suitable arrangements can be used in accordance with the present invention.

The tables in Figs. 2a, 2b, 2c, and 2d are part of system base 206 and store channel driver information and channel driver parameters. The tables of Figs. 2a and 2b store the general information for a channel driver, such as channel drivers 120A, 120B, and 120C, and can be used by any customer support center configuration. The typical data stored in these tables are the file name of the channel driver DLL, the media type of the associated communication channel 130 (e.g. email, fax, etc.), a media string which is used by communication server 109 at run time to invoke a media service for the channel driver, the complete list of channel driver parameters, and the default value for each channel driver parameter. The parameters INBOUND_FLG and OUTBOUND_FLG of table CNCTR (Fig. 2a) indicate whether the channel driver 120 supports inbound and/or outbound communications.

Customer support centers can establish configurations that define the groups of agents that have similar requirements to communicate, therefore requiring access to the same communication channel 130. For example, salespersons within a company may need the ability to communicate via wireless access protocol, whereas telephone operators may not. A configuration can be established for each group within the company. A channel driver profile allows more than one customer support center configuration to share a single channel driver 120, with each additional channel driver profile overriding the values of some channel driver parameters such as the location of the channel driver DLL. For example, due to the network architecture of the company, salespersons for the company in Phoenix may use a different channel driver 120 than salespersons in Palo Alto. A channel driver profile will enable the Phoenix and Palo Alto salespersons to use the same channel driver but point to different DLLs. The term channel driver 120 is used herein to include at least one channel driver profile providing default values for the channel driver parameters.

The tables in Figs. 2c and 2d store the channel driver profile for a particular customer support center configuration and the channel driver profile is not shared or used by other customer support center configurations. Typically, an administrator uses the table CNCTR_PARM to override a default value for a channel driver parameter for the particular customer support center configuration. Referring to Fig. 2a, the string stored in the variable CNCTR_MEDIA_STR is based on a list of names of communication media supported by the channel driver 120. An administrator enters the name of the media in the CNCTR_MEDIA_STR field in character string format. The string stored in this field is used to determine the channel driver 120 to issue a command or from which an event originated. If one channel driver 120 supports multiple types of communication media, the administrator creates one record for each media type. The following examples show the parameters in the CNCTR table for telephone, email, and web chat media:

{"XYZ Phone Driver", "Telephone", "xyz.dll", "Y", "Y", "XYZ Phone Implementation",, "N"},

{"XYZ Email Driver", "Email", "xyz.dll", "Y", "Y", "XYZ Email Implementation",, "N"},

{"XYZ Web Chat Driver", "Web Chat", "xyz.dll", "Y", "Y", "XYZ Web-Chat Implementation",, "N"}

Note that when a work item is submitted to UQ system 102 (Fig. 1A) for agent assignment, the CNCTR_MEDIA_STR is also passed with the work item to help UQ system 102 to identify an agent with skills in using that media type.

An example of an algorithm for determining the list of channel drivers 120 for a particular agent is as follows:

1. Determine the configuration ID for the agent by searching AGENT table (Fig. 2j).

2. For the configuration ID, search the CFG_PROF table (Fig. 2o) for the list of channel driver profiles associated with the configuration.

3. For each of channel drivers 120, load the channel driver information and channel driver parameters from CNCTR, CNCTR_PARM, PROF, and PROF_PARM tables (Figs. 2a-2d, respectively).

An example of an algorithm for loading a list of channel drivers 120 upon the agent logging in to client/server system 100 is as follows:

1. For each of channel drivers 120,

a. If the DLL has not loaded, load the DLL

b. Pass the channel driver parameters and ask the channel driver for the handle of a driver object.

c. Request the handle of a service object by passing the media type of the channel driver identified in CFG_PROF (Fig. 2o) as being associated with the agent.

2. End loop

By default, an agent is authorized to access all channel drivers 120 associated with the configuration to which the agent belongs. For example, if the agent belongs to "Customer support center 1," all channel driver profiles configured in "Customer support center 1" are accessible for all agents in "Customer support center 1" by default. The administrator can

further limit the agent's access to channel drivers 120 using table AGENT_LIM (Fig. 2m) that lists the channel driver profiles the agent cannot access.

Agent preferences are stored in table AGENT_PREF (Fig. 2e) in a centralized database so that an agent's settings are available independently of the type of client or communication channel being used. A user interface for modifying the settings is also supplied in an embodiment of the present invention.

Embodiments of the present invention support multiple communication media channels and agent assignment with UQ system 102 (Fig. 1A). Table AGENT_STAT (Fig. 2f) stores the current working status of a particular agent for all types of media that the agent is authorized to use. From this table, the administrator can see a list of media types that agent is currently authorized to access and the status of each media type.

When the "NOT_READY_FLG" parameter in table AGENT_STAT (Fig. 2f) indicates that an agent is not ready to take work items, UQ system 102 (Fig. 1) will not assign any work items to the agent. The "BUSY_FLG" parameter indicates that the agent is busy.

Table AGENT_STAT is updated mainly at run time. When the agent first logs on using the user interface, one record for each media type that the agent is authorized to access is created. For example,

{"agent_emp_id", "Phone Control", "", "", "1234", ""},

{"agent_emp_id", "Email/Fax", "", "", "1234", ""},

{"agent_emp_id", "Web Chat", "", "", "1234", ""}

The records are updated according the agent's working status. For example

{"agent_emp_id", "Phone Control", "Y", "", "1234", "Y"} indicates that agent is not ready but is talking on the phone,

{"agent_emp_id", "Email/Fax", "Y", "", "1234", ""} indicates that the agent is not ready to accept Email/Fax type of work, and

{"agent_emp_id", "Web Chat", "N", "", "1234", "Y"} indicates that the agent is ready to accept web chat type work and he or she is currently working on a web chat session.

Referring to table MEDIA_STAT (Fig. 2d), the parameter "MEDIA_OBJECT_STR" for phone is the agent's extension number. For email, it is the mailbox name or the sender's email address. For fax, it is the fax number. The form of the content of MEDIA_OBJECT_STR is defined in each of the channel drivers 120.

"WORKING_SINCE_DT" is the time the agent starts to talk on the phone, or the time the agent starts to work on a work item such as an email or fax.

"WORK_ITEM_STR" is the unique string to identify the work item and the value of the field is determined by communication server 109. The MEDIA_STAT table is updated at run time to reflect the agent's current work status. An example of an agent's data records at run time is as follows:

{"agent_id", "Phone Control", "Ext. 5216", "6/25/2000 12:34:45", "phone_item_str", "1-1S2-X7E"},

{"agent_id", "Email", "info@company.com", "6/25/2000 11:34:00", "email_item_str", "1-1S2-X7D"}

The above records show that the agent is currently talking on extension 5216 and is working on an email sent to info@company.com.

Multiple extensions and multiple queues are supported in client/server system 100 (Fig. 1) using tables TELESET, EXTENSION, and AGENT_QUE, Figs. 2h, 2i, and 2j, respectively. The following terms are referenced in Figs. 2h, 2i, and 2j. The term automatic call distribution (ACD) extension refers to a type of extension that is used to log in to an ACD queue associated with an ACD switch such as ACD switch 130E . Once an extension logs in to the ACD queue, the ACD switch begins to dispatch customer calls to the extension. One ACD extension can log in to one or more ACD queues.

The term standard extension refers to a type of telephone extension that is not allowed to log in to the ACD queue. Standard extensions are mainly used for dialing outbound calls

or answering internal calls. The ACD switch does not dispatch customer calls to a standard extension.

The term agent ID refers to an identifier used by client/server system 100 to identify the agent. In order for client/server system 100 to be aware of the agent's availability, each customer support center agent is assigned an agent ID. When the agent logs in to a communication channel having an ACD switch 130E, the agent ID is provided to the ACD switch 130E. Depending upon the configuration of the system, either the ACD switch 130E or UQ system 102 determines an available agent ID for the work item. Then either the ACD switch 130E dispatches the customer call to the ACD extension of the agent ID or, when UQ system 102 is used to assign agents, communication server 109 uses one of channel drivers 120 to dispatch the customer call to the ACD extension of the agent ID.

"Multiple DN" refers to multiple extensions configured for one telephone handset, and one or more extensions are ACD extensions.

"Multiple queue" means that one ACD extension can log in to multiple queues. In general, since an ACD queue is a list of agent IDs, as long as the agent ID is acceptable for ACD queue, any ACD extension can be used to login to ACD queue.

In one embodiment, a method for determining the list of extensions for an agent includes searching by the agent's ID in the AGENT table (Fig. 2j) to find the primary Teleset ID in the ACTIVE_TELESET_ID parameter, which identifies the primary handset used by the agent. The extension list is then determined from the DN_EXT parameter in the EXTENSION table (Fig. 2i). Once the list of extensions is found, all extensions that the agent uses can login to all ACD queues defined in the AGENT_QUE tables (Fig. 2l) for that particular agent.

As described above, customer support centers can establish configurations that define the groups of agents that have similar requirements to communicate, therefore requiring access to the same communication channel 130. Configuration base 202 includes tables about configurations. CFG table (Fig. 2n) contains information about configurations. Configuration data includes a configuration name and an INGRP_FLAG indicating whether this configuration is for inbound response groups used in inbound communication receiver 170. CFG_PROF table (Fig. 2o) is the configuration / channel driver profile relationship

table showing which channel driver profiles belong to each configuration. Each configuration has a single channel driver profile.

AGENT_CFG table (Fig. 2p) is the agent / configuration relationship table showing which agents belong to each configuration.

CFG_PARM table (Fig. 2q) is the configuration parameter table. A name and a value are provided for each configuration parameter. An ACTIVE_FLG field is a flag indicating whether the value of the configuration parameter is active.

The command and event data structure 204, includes information describing commands and events implemented by channel drivers 120. This information includes associating each command with a channel driver 120 and each event with a channel driver 120.

CMD table (Fig. 2r) includes commands for each channel driver 120. As described above, a vendor providing a channel driver 120 specifies the commands that it supports. A command is issued to channel driver 120 by communications server 109 to perform a command using communication channel 130. Every click on a button of toolbar 105 invokes a command, which is issued to channel driver 120.

A command can have a group of associated commands which operate as subcommands. A group command includes other commands with a Subcommand keyword.

Following is an example of a single command for making a telephone call to a contact.

```
[Command: MakeCalltoContact]                                  Command definition
    CmdData              =    "MakeCalltoContact" Command parameter
    DeviceCommand        =    "MakeCall"          Command parameter
    Description          =    "Make Call to Contact" Command param.
    Hidden               =    TRUE                Command parameter
    [CmdData: MakeCalltoContact]                  Command data def.
        BusComp          =    "Contact"           Command parameter
        RequiredField.'Work Phone #'      ="?*"    Command parameter
        Param.PhoneNumber =    "{Work Phone # : Lookup}  Command
                                                   Parameter
```

Following is an example of a group command for making a telephone call to a contact:

```
[Command: MakeCallGroup]
        Hidden          =   TRUE
        SubCommand      =   MakeCalltoPhone
        SubCommand      =   MakeCalltoSRContact
        SubCommand      =   MakeCalltoSROwner
        SubCommand      =   MakeCalltoEmployee Home
```

The following example command can be either a single command or a subcommand.

```
[Command: MakeCalltoPhone]                          Command definition
        CmdData         =   "MakeCalltoPhone"       Command parameter
        DeviceCommand   =   "MakeCall"              Command parameter
        Description     =   "Make Call to {@Phone}" Cmd param
        Hidden          =   TRUE                    Command parameter
        [CmdData: MakeCalltoPhone]                  Command data def.
        [CmdData: MakeCalltoPhone]                  Command data def.
                RequiredField.'Work Phone #'    ="?*"
Param.PhoneNumber =     "{@Phone:   PhoneTypeLookup}
```

A command can have a command data section with a CmdData keyword to specify the data parameter in order to communicate with channel driver 120.

When a customer support center configuration includes multiple channel drivers 120, it is then possible for communication server 109 to determine which commands and events are handled by each of channel drivers 120. This configuration can also help distinguish between channel drivers 120 from different vendors that use the same name for commands performing different functions.

Following is an example of a command with a data section having a CmdData keyword.

```
[Command: MakeCalltoContact]
        CmdData         =   "MakeCalltoContact"
        DeviceCommand   =   "MakeCall"
        Description     =   "Make Call to Contact"
        Hidden          =   TRUE
        [CmdData: MakeCalltoContact]
                BusComp         =   "Contact"
                RequiredField.'Work Phone #'    ="?*"
```

Param.PhoneNumber = "{Work Phone # : Lookup}

The event table contains events that are sent to communication server 109 from channel driver 120. Vendors specify the events that will be sent in channel driver 120. An event response determines how communication server 109 reacts upon receiving each media event. The process of handling an event includes the following: searching for the event handler for the event, querying a customer support center database to determine the appropriate event response, and logging the event.

An example of an event, the event handler, event response, and event log for an InboundCall event are shown below:

```
[EventHandler:OnInboundCall]                    first stage, EventHandler
definition
  DeviceEvent      = "EventRinging"                media event definition
  Response         = "OnInsideCallReceived"    EventResponse declaration
  Filter.Call      = "?*"                      EventHandler parameter
  Order            = "1"                       EventHandler order
[EventResponse:OnInboundCallReceived]           second stage, EventResponse
definition
  QueryBusObj      = "Contact"                 EventResponse parameter
  QueryBusComp     = "Contact"
  QuerySpec        = "'Work Phone #'='{ANI}'"
  SingleView       = "Service Contact Detail View"
  MultiView        = "Contact List View"
  FindDialog       = "Service Request"
  FindField.CSN    = "Ask Caller"
  FindLog          = "LogIncomingCallContactNotFound"      EventLog
declaration
  SingleLog        = "LogIncomingCallContactFound"     EventLog declaration
  Log              = "LogIncomingCallContactNotFound"      EventLog
declaration


[EventLog:LogIncomingCallContactFound]          ß EventLog definition
  Display          = "TRUE"                    ß EventLog parameters
  BusObj           = "Action"
  BusComp          = "Action"
  LogField.Type    = "Call – Inbound"
  LogField.'Account Id'  = "{Contact.'Account Id'}"
  LogField.'Contact Id'  = "{Contact.Id}"
  LogField.Description   = "Inbound call"
  LogField.'Call Id'       = "{ConnID}"
```

AfterCall.'ACD Call Duration'= "{@CallDuration}"

Each event handler corresponds to an event provided by channel driver 120 and it is sequenced among the event handlers for an event. Each event handler has an event response. An event response can be shared among event handlers. An event response can have multiple event logs, and an event log can be shared among event responses.

When operating in session mode, communication server 109 is under the control of session mode communication server 110. Session mode communication server 110 receives incoming events such as customer support requests and communicates interactively with the agent by controlling a user interface presented to the agent. Preferably the incoming customer support request is communicated to the agent at substantially the same time the customer support request is received by the communication channel 130, with brief intermissions only to allow for processing and transport time in transporting the customer support request. This ensures that the customer's waiting time is minimized, particularly for requests for live interaction with an agent.

When an event such as arrival of an incoming telephone call occurs, the user interface notifies the agent using a notification function to change the user interface to capture the agent's attention. For example, a notification function can cause a button to blink to notify the agent of the phone call. A notification function can also display other information such as information about the caller before the agent picks up the phone. When the agent uses toolbar 105 to accept a telephone call, put a call on hold, or release a call, the user interface sends a command to session mode communication server 110, which communicates with one of channel drivers 120 to issue the command to the communication channel controlling the telephone.

Fig. 1B shows a detailed view of one embodiment of session mode communication server 110. Session mode communication server 110 maintains knowledge of clients 104 to which it is connected, here web browser client 104A. When a communication from communication channel 130, here ACD switch 130E is received, communication server 109 dispatches the request to the appropriate server component in client/server system 100 for execution.

Session thread 122 represents a session during which an agent interacts with client/server system 100 using web browser client 104A. A customer uses a customer communication device, here a telephone, to access the communication channel. The agent also uses a communication device, such as a telephone headset, to access the communication channel.

Session thread 122 listens for inputs from its web browser client 104A and dispatches notifications of events from ACD switch driver 120 to web browser client 104A. Session thread 122 uses a communication channel manager such as communication channel manager 124 to interact with a ACD switch driver 120. Each channel driver 120 provides an active connection such as active connection 133 between the client and the associated communication channel. Channel driver 120 can be implemented to establish a persistent connection for interactive communication between client 104 and communication channel 130E but providing a persistent connection is not required by communication API 125.

The following examples describe processes that are followed by web browser client 104A during startup, initialization and operation. The processes for web browser client 104A are applicable to other types of clients, as will be explained in further detail below.

When web browser client 104A begins execution:

1. Web browser client 104A downloads program instructions for generating a user interface on the display for the web browser, such as toolbar 105, shown here for implemented using Java applet 116, from web server 188. Java applet 116 also establishes persistent HTTP connection 131 between Java applet 116 and web server 188 so that web server 188 can continuously provide information to web browser client 104A.

2. Web browser client 104A interfaces with session mode communication server 110 via web engine session thread 166. Object manager 107 spawns web engine session thread 166 to interface with web browser client 104A using web engine plug-in 185 and web engine 115. Communication client service 160 provides all communication related to the user interface with web browser client 104A.

3. Communication client service 160 requests the object manager 107 for communication service. Communication service 113, which provides all communications not related to the user interface, is provided.

4. Communication service 113 loads configuration information such as commands, events, agent information and preferences, channel driver information and channel driver parameters.

5. Communication service 113 registers an asynchronous event receiving function with object manager 107 to be invoked when an asynchronous event is subsequently received. The asynchronous event receiving function is also referred to as a callback function. Receiving asynchronous events is described in further detail below.

6. Communication service 113 request an active connection 135A between object manager 107 and web engine plug-in 185 and an active connection 135B between communication service 113 and session mode communication server 110. Persistent HTTP connection 131, and active connections 135A and 135B enable session mode communication server 110 to continually push user interface changes to toolbar 105 using Java applet 116.

7. Session mode communication server 110 spawns a session thread such as session thread 122 in response to the connection request.

8. Session thread 122 runs communication channel manager 124.

9. Communication channel manager 124 loads ACD switch driver 120D and passes the channel driver parameters determined by communication service 113.

10. ACD switch driver 120D establishes an active connection 133 to the ACD switch 130E. A vendor implementing channel driver 120 may choose to provide a persistent connection to the communication channel 130, as for telephone connections such as active connection 133. However, a persistent connection is not required by communication API 125.

When the agent performs an activity using web browser client 104A that requires a command to be executed, such as clicking a button on toolbar 105:

1. Communication client service 160 searches the command configuration data previously loaded for the command to invoke. It also collects the data associated with that command and then passes the command and data to communication service 113.

2. Communication service 113 passes the command and data to communication channel manager 124.

3. Communication channel manager 124 then determines which of channel drivers 120 performs the command requested by the client, and passes the command and data to the channel driver 120 such as ACD switch driver 120D for execution.

4. ACD switch driver 120D issues the command to the communication channel 130. In this example, the ACD switch driver 120D issues the command to ACD switch 130E.

When a channel driver 120 such as ACD switch driver 120D needs to push an event (status data or an incoming event such as a customer call) to web browser client 104A:

1. ACD switch driver 120D receives the event and posts the event to communication channel manager 124. This requires asynchronous interruption at session thread 122 for event posting.

2. Communication channel manager 124 pushes the event to communication service 113.

3. Communication service 113 receives the event and executes the registered asynchronous event receiving function.

4. The registered asynchronous event receiving function inserts the event sent from ACD switch driver 120D into an event queue stored inside object manager 107.

5. A frame manager (not shown) running in session thread 122 picks up the event from the event queue and invokes the registered asynchronous event receiving function using communication client service 160.

6. Communication client service 160 asks communication service 113 to process the event.

7. After communication service 113 has processed the event, communication client service 160 continues to communicate with Java applet 116 to control the web browser for user interface changes.

Fig. 1C shows components included in one embodiment of request mode communication server 140. Request mode communication server 140 handles the distribution of information via communication channels according to the request. An example of the operation of request mode communication server 140 is session mode communication server 110 sending a request to request mode communication server 140 to send a large number of emails on its behalf. This enables session mode communication server 110 to devote its resources to controlling the user interface, issuing commands, and handling events.

A request mode server thread such as server thread 142 is spawned when request mode communication server 140 begins execution. Communication manager 152 is loaded to collect data for the request. Request mode communication server 140 determines the appropriate channel driver to handle the request and directs a communication channel manager 156 to load email driver 120E. Communication channel manager 156 dispatches the request and data to email driver 120E, which sends the information to email communication channel 130F. In the embodiment shown in Fig. 1C, email driver 120E sends the emails via email server 132 to email client 134.

As another example of the operation of request mode communication server 140, object manager 107 can send one or more work items from UQ system 102 to request mode communication server 140. Similar to the previous example, a request mode server thread is spawned and communication manager 152 is loaded to collect data for the request. Request mode communication server 140 determines the appropriate channel driver to handle the request and directs a communication channel manager 156 to load an appropriate driver, such as email driver 120E. Communication channel manager 156 dispatches the request and data to the driver, which sends the information to a communication channel.

Fig. 1D shows an example of one implementation of inbound communication receiver 170. One embodiment of inbound communication receiver 170 is designed to serve inbound customer support requests with no connection to or knowledge of a client. This contrasts

with session mode communication server 110, which communicates with a client to provide a user interface to at least one agent. In one implementation, inbound communication receiver 170 handles customer support requests that can be held in a queue for future processing, such as fax and email, whereas session mode communication server 110 handles high priority support requests that should be processed as quickly as possible, such as telephone calls, to improve customer response time. In another implementation, both inbound communication receiver 170 and session mode communication server 110 can handle high priority support requests.

Inbound communication receiver 170 uses channel drivers 120 such as email/fax channel driver 120F to "listen" for particular types of customer support requests from a common source. Email channel driver 120F handles all email messages directed to a particular email address and all faxes sent to a particular fax number. To avoid overlap among agents, inbound communication receiver 170 can be configured to work with UQ system 102 to assign an agent to the inbound customer support request (email 173 or fax 175) and route the customer support request to a component associated with or representing the assigned agent, such as a client.

Inbound communication receiver 170 is also configured during initialization to recognize events, such as receiving a customer support request, and to include corresponding channel driver information and background profiles to handle recognized events. Background profiles include one or more monitored media objects, such as a list of email addresses, fax numbers, and web-chat end points. For example, email communication channel 130G represents a background profile for info@company.com and fax communication channel 130H represents a background profile for fax number 1-800-123-4567.

Inbound communication receiver 170 spawns a server thread such as server thread 174 to handle inbound events, such as customer support requests. This contrasts to session mode communication server 110, which spawns a session thread such as session thread 122 for each client 104 being used by an agent. Communication channel manager 177 then initializes a service such as fax service object 183A, email service object 183B, or phone service object 183C with the designated background profile.

When the email/fax channel driver 120F receives an incoming customer support request, e.g. new fax 175, fax channel driver 120F posts the event to communication channel manager 177. This posting interrupts the idle state of server thread 174 and causes server thread 174 to invoke communication channel manager 177 to process the event. Communication channel manager 177 determines how to respond to the event based on an event response included in an event response table, such as EVTRESP (Fig. 2y), and invokes the appropriate media service, such as fax service object 183A. If the event response also specifies notifying UQ system 102 of the event, the event is then passed to UQ system 102 via UQ business service 106. A response to the event notification is returned to inbound communication receiver 170 via UQ business service 106.

In alternative embodiments, client/server system 100 can support multiple types of clients 104 having hardware/software configurations that are different from web browser client 104A. Fig. 1E shows an alternative embodiment of client/server system 100 that supports web browser client 104A, thin client 104B, and dedicated client 104C.

Thin client 104B includes one or more client software modules that are installed and executed on the client computer system used by the agent. Thin client 104B provides minimal functionality, with the majority of the functions for thin client 104B are performed by application server 126. It is often desirable to use thin clients so that application programs can be updated once in a centralized location instead of multiple times for each thin client 104B.

Thin client 104B provides more functionality on the client side than web browser client 104A, and can, for example, perform some functions of object manager 107. Thin client 104B also controls the user interface including toolbar 105. If changes are necessary to the functions performed on the client side, a new copy of thin client 104B must be installed on each individual agent's computer system.

Dedicated client 104C includes software modules that perform a significant portion of the functions required to support an agent. Dedicated clients are sometimes referred to as "fat clients," in contrast to the "thin client" designation. If changes are necessary to the functionality provided by dedicated client 104C, a new copy of the dedicated client software modules usually must be installed on the client computer system.

Dedicated client 104C provides even greater functionality than does thin client 104B, including, for example, all functionality provided by object manager 107, web server 188, communication client service 160 (Fig. 1B), and communication service 113. Because dedicated client 104C assumes all responsibility for the user interface and toolbar 105, there is no communication between dedicated client 104c and communication server 109, web server 188, web engine plug-in 185 and web engine 115 (Fig. 1B). Dedicated client 104C does include web server 149 that is capable of interfacing with UQ system 102, and object manager 151 to communicate with channel drivers 130.

It is important to note that other types of clients having hardware and software components that are different from clients 104A, 104B, and 104C can also be integrated with client/server system 100.

Communication API

Referring now to Figs. 1F-1J, communication API 125 is provided in one embodiment of the present invention for channel drivers 120 to communicate with communication server 109. Note that communication server 109 is used in the following discussion of communication API 125 to represent session mode communication server 110, request mode communication receiver server 140, or inbound communication receiver 170.

As shown in Fig. 1F, an example of communication between communication server 109 and channel driver 120 using communication API 125 includes three types of objects: driver objects 189, service objects 183, and client objects 179. Driver objects 189 and service objects 183 are instantiated at the channel driver 120, however client objects 179 are instantiated at communication server 109. Communication server 109 interfaces with driver objects 189 and service objects 183, but only service objects 183 communicate with client objects 179.

Driver objects 189 maintain the instantiation of service objects 183. Any special steps for constructing and destructing service objects 183 can be implemented in driver objects 189. Multiple driver objects 189 can be included to manage different types of media. Also, a single driver object 189 can manage one type of service objects 183 or different type of service objects 183. For example, a single driver object 189 can manage phone, email and fax media.

As an example of the operation of driver objects 189, when communication server 109 is starting up, the channel driver 120 data link library (DLL) is loaded. Communication server 109 calls CreateISCSDriverInstance() function of channel driver 120 to ask for the construction of a driver object 189. The channel driver 120 returns the driver handle back to communication server 109. The channel driver 120 determines how driver objects 189 are created. If driver objects 189 already exist, for example, the channel driver 120 could simply pass the handle of an existing driver object 189 instead of creating a new one.

Service objects 183 provide functionality in the form of device commands to interact with the associated media type. For example, making an outbound call, or sending an outbound email is implemented at service objects 183. A service object 183 is usually associated with a single type of media. For example, there can be service objects 183 for phone media and other service objects 183 for email media. Communication server 109 interfaces directly with service objects 183 to invoke a device command.

After communication server 109 obtains the handle to a service object 183, communication server 109 will use the service handle directly to interact with the service object 183. Since service objects 183 are created by driver objects 189, service objects 183 can inherit some facilities from driver objects 189 and/or share some resource with driver objects 189. For example, driver objects 189 can establish and maintain the physical TCP/IP connection to a middleware server of a communication channel 130 and service objects 183 can share the connection with the driver objects 189.

After communication server 109 obtains the driver handle, communication server 109 uses a RequestService() function to request a service object 183 for the specified media type. The driver returns the handle of the corresponding service object 183 to communication server 109. Communication server 109 then uses this handle in an InvokeCommand() function directly to request the corresponding service object 183 for executing a particular type of function.

Client objects 179 are instantiated and implemented by communication server 109. The handles to client objects 179 are passed to service objects 183. Service objects 183 can utilize the client handles and invoke the function to be executed at communication server 109.

Every service object 183 will have its corresponding client object 179. Therefore, each client object 179 has knowledge of the media type that its corresponding service object 183 is using. Since service objects 183 can each be instantiated for different media from different driver DLLs, this one-to-one relationship allows a client object 179 to know the channel driver 120 and service object 183 that initiate the notification when client object 179 receive notification from service object 183.

Fig. 1G shows an example of an architecture for driver object 189 instantiated by channel driver 120. Driver object 189 creates three service objects 183A-1, 183A-2, and 183A-3 of the same media type, such as email. Each service object 183A-1, 183A-2, and 183A-3 has its own dedicated client object 179A-1, 179A-2, and 179A-3, respectively.

Fig. 1H shows an alternative architecture for driver object 189 that creates three service objects 183A, 183B, and 183C for different types of media. Each service object 183A, 183B, and 183C has its own dedicated client object 179A, 179B, and 179C, respectively, for processing events with the corresponding media type. An example of this architecture is shown in Fig. 1D for inbound communication receiver 170 that includes client object 179A for handling fax media, client object 179B for handling email media, and client object 179C for handling phone media. Client objects 179A, 179B, and 179C correspond to fax service object 183A, email service object 183B, and phone service object 183C, respectively.

Fig. 1I shows two driver objects 189A, 189B instantiated in the channel driver 120. Each driver object 189A, 189B is designated for a different middleware server of communication channel 130 and includes resources specific to the type of middleware server. For example, driver object 189A may use a TCP/IP connection to Middleware Server A and driver object 189B may have a direct connection to Middleware Server B. The service objects 183 created under each driver object 189A, 189B are specific to the middleware server with which the driver object 189A, 189B is associated.

There are several alternatives for implementing asynchronous notification of events from middleware servers to driver objects 189 including:

1. Traditional TCP/IP socket. The driver objects 189 connect to the TCP/IP port of a middleware server. Events are sent through TCP/IP connection.

2. OLE interface. One example is the IAdviseSink interface in OLE.

3. Any other inter-process communication scheme.

With alternative 1, since the driver objects 189 are implemented as a DLL, the driver object DLL either constructs a listening thread which blocks on select() call until the arrival of an event, or a polling thread which periodically polls the middleware server for the arrival of event. Polling threads are useful for low-priority media type, e.g. email or fax, because polling periods typically last seconds or minutes. Polling threads are not as useful to detect high-priority media events, such as phone requests, because it is desirable to report the arrival of an incoming call at any time. Listening threads generate less network traffic than polling threads, and are generally useful for high priority and low priority media, however, some types of middleware servers do not support listening threads.

To implement both polling threads and listening threads, a "task" thread is required in the driver object DLL. The "task" thread can be executed in driver objects 189 as shown in Fig. 1J or in service objects 183 as shown in Fig. 1K.

Referring to Fig. 1J, a task thread (or listen thread) implemented the driver objects 189 may be "shared" by all service objects 183. For example, this listen thread can listen for all incoming events for all service objects 183. Once the listen thread receives an event, the listen thread then invokes and executes the event handling function implemented at service objects 183.

Referring to Fig. 1K, if the listen thread is implemented at the domain of service objects 183, every service object 183 constructs its own listen thread and the listen thread is not shared. Each listen thread is listens to a different target. For example, listen thread for user 1 listens for events on the first phone extension (ext. 1234), while the listen thread for user 1 listens for events on the second phone extension (ext. 5678).

Client objects 179 are a collection of function pointers implemented by Communication server 109 and passed to the service objects 183 for asynchronous event notification. In one implementation, when the listen thread in channel driver 120 receives an event, the following processes occur:

1. Service object 183 calls HandleEvent(). HandleEvent implemented in corresponding client object 179 is executed.

2. Client object 179 queues this event to a memory cache.

3. Client object 179 interrupts or signals the server thread 174 (Fig. 1D) for Communication channel manager 177 to indicate the arrival of an event. Once this process is completed, the listen thread waits for the next event.

4. During the next cycle of server thread 174, main thread sees an event is available in the memory cache. It dequeues the event out of the memory cache and continues the processing.

Communication API Commands

Communication API 125 includes commands and data structures to allow third parties to develop applications that can integrate with client/server system 100. The data structures include arrays for passing data elements such as an agent's key value element, key value parameters, and string parameter lists.

The following provide examples of runtime status flags that can be used in communication API 125:

```
NOTSUPPORTED  = 1;  Command is not supported
DISABLED      = 2;  Command is disabled at this time
CHECKED       = 4;  Command is in "checked" state, for example when agent
is in busy mode the "busy" command will be "checked"
BLINKING      = 8;  This is special effect flag to enable the blinking "answer
call" command
NOPARAMSOK    = 16; Command does not require any parameters to execute
STRPARAMSOK   = 32;
```

Command can be executed by providing single unnamed string parameters. Such commands are invoked when the agent types something in the edit control of the communication toolbar 105 and clicks the corresponding button.

The following provide examples of commands that can be used in one embodiment of communication API 125:

The MediaType command is used from channel driver 120 to provide the media type. The media-type-string is passed to the channel driver 120 at CreateISCDriverInstance().

PHONECONTROL = 1
CALLROUTING = 2
EMAIL = 3
FAX = 4
WEBCALL = 5
WEBCHAT = 6

Channel driver 120 uses the CommandTypeEx function to request different services, such as making calls and sending messages, from communication server 109.

The SCObjectype function is used to monitor the communication objects, which can be represented by the following parameter values:

OB_LINK = 1
SWITCH = 2
QUEUE = 3
TELESET = 4
DN = 5
AGENT = 6
CALL = 7
CALLROUT = 8
EMAIL = 9
FAX = 10
WEBCALL = 11
WEBCHAT = 12
OTHERS = 1000

The function ObjectProperty can be used to provide properties of monitored communication objects, such as:

OP_ONOFF = 1
OP_AGENTID = 2
OP_NOTREADY = 4
OP_BUSY = 5
OP_DESCRIPTION = 7
OP_TIMEINQUEUE = 9
OP_QUEUEID = 12
OP_ISLOGON = 13

Channel Driver Functions

In one embodiment, a driver objects 189 within each of channel drivers 120 include the following functions:

FreeSCStrParamList is invoked by communications server 109 to release the memory which is initially allocated by channel drivers 120.

RequestMediaTypeList is invoked by communications server 109 to query the list of media type strings supported by channel drivers 120. It can include the parameter mediaTypeList, which is a list of media-type strings.

RequestCommandEventList is invoked by communciations server 109 to query the list of device commands and device events supported by the channel drivers 120.

FreeSCStrParamList() is invoked by communication server 109 to release memory.

RequestCommandEventList is invoked to generate lists of commands and events that are implemented for a particular media type. The parameters can include an input parameter specifying the media type, and output parameters that include lists of the commands and events.

CreateISCDriverInstance is invoked to create a channel driver 120. The following parameters can be used:

mediaTypeStr: the media-string that is defined by a particular driver implementation.

languageCode: the language string, e.g. "ENU" for English, "FRA" for French, "DEU" for German, "PTB" for Portuguese-Brazilian, "ESN" for Spanish, "ITA" for Italian, and "JPN" for Japanese.

connectString: the connect string for the channel driver 120

datasetParams: the parameter list collected from the configuration

handle: the handle to channel driver 120 returned by the channel driver 120

RequestService requests media functions from the channel driver 120. The following parameters can be used:

clntInterface: the interface at the client side

connectString: the connect string for the service objects

datasetParams: the parameter list collected based on the configuration

serviceHandle: the handle to the service objects returned by the driver

ReleaseISCDriverInstance is invoked by communication server 109 to release the driver instance specified by the driver handle supplied as a parameter.

Service Object Functions

In one embodiment, service objects 183 within each of channel drivers 120 can include the following functions:

ReleaseISCServiceInstance is invoked to release the service object's handle.

NotifyEventHandlingFinished is invoked by communications server 109 to notify the channel driver 120 that the event handling is complete and the channel driver 120 can move on or continue the process. This function is invoked to respond to HandleEvent's notifyWhenDone parameter. The following parameter list can be used:

Handle: identifier of the service object

trackingID: an identifier for the work item for which the communications server 109 was doing event handling.

result: the result of event handling query of the list of media type strings supported by the channel driver 120.

InvokeCommand is invoked by communications server 109 to invoke a driver command. The following parameter list can be used:

Handle: identifier of the service object

clntCmdTrackID: the unique ID for the InvokeCommand request

name: the command name to invoke

stringParam: the string from "Phone #" edit box on the toolbar 105

datasetParam: the parameter list collected based on the configuration

InvokeCommandEx is invoked by communications server 109 to invoke a certain type of command. The following parameter list can be used:

Handle: identifier of the service object

clntCmdTrackID : the unique ID decided by the communications server 109 for this InvokeCommand request

commandType : the type of command the communications server 109 wants to execute

datasetParam : the predefined parameter list set by the communications server

ReleaseWorkItem is invoked by communication server 109 to request release of a work item. Parameters can include:

Handle: identifier of the service object

TrackingID: identifier of the work item.

SuspendWorkItem is invoked by communication server 109 to request the service object to suspend a work item. Parameters can include:

Handle: identifier of the service object

TrackingID: identifier of the work item.

ResumeWorkItem is invoked by communication server 109 to request the service object to resume a work item. Parameters can include:

Handle: identifier of the service object

TrackingID: identifier of the work item.

HandleQueuedEvent is invoked by communication server 109 to pass an event previously queued in UQ system 102 to the service object for handling. The channel driver

120 can treat this as an incoming media event from the middleware server. Parameters can include:

Handle: identifier of the service object

name : the event name (from the original HandleEvent() call)

fields : the event attributes list (from the original HandleEvent() call)

trackingID : the unique ID for the media item

CancelQueuedEvent is invoked by communication server 109 to notify the channel driver 120 that a media-event is cancelled, released, or transferred by UQ system 102. This function is the companion function of HandleQueuedEvent(). The following parameters can be used:

Handle: identifier of the service object

name : the event name (from the original HandleEvent() call)

trackingID : the unique ID for the media item

Client Object Functions

The following are examples of functions that can be included in Client Objects 179. The interface to these functions can be implemented with a function pointer so that driver objects 189 do not need to link to any libraries in communication server 109.

RequestService() issues a request from client objects 179 to driver objects 189. The CLIENT_INTERFACE object and the CLIENT_HANDLE are passed as parameters.

ReleaseClientInstance causes driver object 189 to release a client object's handle.

BeginBatch and Endbatch are designed to saving network overhead. The ISC_CLIENT_INTERFACE function calls between BeginBatch and EndBatch will be cached and sent out at EndBatch call. These two functions can be used at the discretion of the driver object 189. This is the example usage,

```
BeginBatch_Helper(clientInterface);
  CacheCommandInformation_Helper(clientInterface, ...);  <-- cached
; ; ; ; // some processing
if (error)
  HandleError_Helper(clientInterface, ...); <-- cached
HandleEvent_Helper(clientInterface, ...); <-- cached
EndBatch_Helper(clientInterface); <-- All requests will be sent out in one request
*/
```

HandleEvent is used to handle the named event received from the driver, using the given fields. By calling this method, the driver notifies the client objects 179 of the event, such as a call coming in on the monitored teleset. The following is the parameter list:

Handle: identifier of the service object

name : the event name

fields : event attributes list

notifyWhenDone : When set to TRUE, Client objects 179 will use notifyEventHandlingFinished() to notify the driver as soon as the event handling is done.

trackingID : the ID uniquely identifies the work item that this event is associated with, e.g. call ID, email ID or web-chat session ID. The length of trackingID should not exceed MAX_TRACKING_ID_LEN.

ShowStatusText displays textual status information in the status line of the client objects 179. The following is the parameter list:

Handle: identifier of the service object

text : the text to display at the client status bar

HandleError handles asynchronous errors and logs them to an error log file. The following parameters can be used:

Handle: identifier of the service object

clntCmdTrackID : if not 0, it is the same "clntCmdTrackID" value passed to InvokeCommand() to reflect the error caused by the request in InvokeCommand(). If it is 0, the error occurs out of context.

error : the error text.

CacheCommandInformation is used to notify the client objects 179 about command status caching. The following parameters can be used:

commandNames : list of command names

commandDescriptions : list of description text for each command

commandStatuses : list of status(SCCommandFlag) for each command

UpdateObjectInformation is used to notify the client objects 179 about status change of objects. The following parameters can be used:

trackingID : the ID uniquely identify the call that causes this information update

objectType : enum of SCObjectType

objectID : the unique ID for this object. For phone, it is the extension. For email, it is the mailbox. For fax, it is the fax number.

datasetInfo : the list of SCObjectProperty value to update. For example, { {"4", "TRUE"}, {"7", "33"} } (SC_OP_ISNOTREADY is TRUE and SC_OP_TIMEINQUEUE is 33 seconds) where the first key of "3" is SC_OP_ISTALKING and the value is "TRUE". The second key of "6" is SC_OP_TALKINGSINCE and the value if "03/12...."

IndicateNewWorkItem notifies client objects 179 about the arrival of new inbound work item(e.g. call, email or fax). The following parameters can be used:

trackingID : the unique ID to identify this work item

oldTrackingID : if the driver or the middleware supports a facility to change the work item's ID. Use this oldTrackingID to identify the old ID. This is to tell client objects 179, "Here is the new work item, but it originated from the old work item".

WorkItemStarted notifies client objects 179 that the agent has started working on one particular work item. This happens when (1) the agent answers a call and the call is connected, or (2) the agent accepts a email/fax work item. In response, client objects 179 set the work item identified by "trackingID" as the active work item and start tracking this work item. The agent will be treated as talking or working. The start time of this work item will be recorded by client objects 179. The following parameters can be used:

trackingID : the unique ID to identify this work item

oldTrackingID : See the comment of IndicateNewWorkItem()

objectType : the object type

objectID : the media object for this work item. For phone, it is the extension.

For email, it is the mail box.

description : the description of work item which will be displayed at the top of combo box. Driver implementation can use UpdateObjectInformation to change the description of work item.

startTime : the time the work item is started

WorkItemReleased is used to notify client objects 179 that a particular work item is released. This happens when (1) the agent releases a call and the call is disconnected, or (2) the agent completes an email/fax work item. In response, client objects 179 stop tracking this work item and remove this work item. The following parameters can be used:

trackingID : the unique ID to identify the work item that is being released.

stopTime : the time the work item is released/stopped.

CleanAllWorkItens notifies client objects 179 that all work items stored in client objects 179 should be removed.

WorkItemSuspended notifies client objects 179 that a work item is suspended. This happens when (1) the agent puts a call to hold, or (2) the agent suspends an email/fax work item. The driver implementation calls this function when suspension is done. In response, client objects 179 save the working context for this particular work item. The following parameter can be used:

trackingID : the unique ID to identify the work item

WorkItemResumed notifies client objects 179 that a suspended work item is resumed. This happens when (1) the agent unholds a call and the call is retrieved, or (2) the agent resumes an email/fax work item. The driver objects 189 call this function when restoring is complete. In response, client objects 179 restore the working context(screen + work-tracking obj) and set the active work item as the one identified by "trackingID". The following parameters can be used:

trackingID : the unique ID to identify the work item

Note that other functions and parameters can be included in communication API 125 instead of, or in addition to, the functions listed herein.

Fig. 3 shows the processing of commands and events by communication server 109. As described above, session mode communication server 110 controls a user interface presented to an agent for handling work items, and session mode communication server 110 is shown in Fig. 3 interacting with web browser client 104. The user interface is consistent for communication using multiple communication channels of different media types. The following description of processing of events by session mode communication server 110 also applies to request mode communication server 140 and inbound communication receiver 170.

An agent logs in to client / server system 100 by activating a user interface object such as a login object of a user interface indicating that he or she is able to begin providing support for customer support requests. An agent can log in to any communication channel 130 associated with a customer support center configuration to which the agent is also associated. At login, web browser client 104A sends a connection command to session mode communication server 110 communicated through intermediate components (omitted here, as shown by the breaks in the arrows) of application server 126, as described in Fig. 1B.

The result of the connection command is that a session is established between toolbar 105 and session mode communication server 110. The session connection enables session mode communication server 110 to push information from communication channel 130 to toolbar 105. If the communication channel 130 is one that allows agents and customers to communicate interactively such as a live web collaboration session, channel driver 120 is responsible for maintaining the persistent connections within the communication channel 130.

Channel driver 120 is implemented according to communications API 125 to communicate with communications server 109. Communications API 125 requires a vendor providing channel driver 120 for a particular communication channel 130 to implement certain functions and data structures in order to communicate with communications server 109, as described above for Figs. 1A-1K.

One requirement of communications API 125 is that channel driver 120 provide instructions to create a driver object and a service object for communicating with communication server 109. The driver object is specific to the media type of communication channel 130. The driver object creates service objects for communication channel 130, such as email service object 183B for email communication channel 130G and fax service object 183A for fax communication channel 130H of Fig. 1D.

Channel driver 120 monitors communication channel 130 for communication activity, as described above with reference to Figs. 1J and 1K. In Fig. 1J, driver object 189 listens to communication channel 130, and in Fig. 1K, service objects 183A and 183B listen. Whether the listening is performed via driver object or a service object 183 is a decision made by the vendor in developing the channel driver 120.

The service objects 183 implement the functionality for communicating with one or more communication channel 130 such as the handshaking and protocol(s) to send commands to and receive events from the hardware devices and/or software elements of communication channel 130.

Upon agent login, session mode communication server 110 loads all channel drivers 120 for the configuration to which the agent using client 104 belongs. A listen thread of session mode communication server 110 then listens to web browser client 104A for commands and the channel driver driver objects 189 or server objects 183 listen for events from channel driver 120 indicating activity on communication channel 130.

When an agent activates a user interface object (such as by clicking on an accept work item button) on toolbar 105, an InvokeCommand function of the user interface object is activated that sends the name of a command to be issued to session mode communication server 110. Session mode communication server 110 determines a channel driver 120 to issue the command by using the command name received from the user interface object to query customer support center database 330. The command table CMD (Fig. 2r), the channel driver table CNCTR (Fig. 2a), and the configuration table CFG (Fig. 2n) are examples of tables that can be used by session mode communication server 110 to determine the channel driver 120 associated with the command. Session mode communication server 110 obtains the parameters necessary for the command from a command parameter table such as CMD_PARM (Fig. 2s) and uses the service objects 183 to provide the command and the parameters to channel driver 120. Channel driver 120 issues the command to the communication channel 130.

When an event from channel driver 120 is received, session mode communication server 110 determines the channel driver 120 for the communication channel 130 that originated the event by querying customer support center database 330. Tables such as channel driver table CNCTR (Fig. 2a), event table EVT (Fig. 2t), and configuration table CFG (Fig. 2n) are among the tables used to identify the channel driver 120.

Having identified channel driver 120 as responsible for originating the event, session mode communication server 110 determines an event response to be made. The event response may be in the form of a data window presented via web browser client 104 as

directed by Java applet 116. Other types of event responses include presentation of a scripted dialogue of questions for the agent to ask the customer, running a software program to perform an operation, calling a business service of a server component of system 100 such as UQ business service 106, and creating a database record in customer support center database 330. An event response corresponds to an event. Event responses are configurable by an administrator using configuration user interface 340 and are stored in an event response table such as EVTRESP (Fig. 2y). Session mode communication server 110 also logs the event response for tracking purposes in an event log table such as EVT_LOG (Fig. 2aa).

Communications server 109 uses configuration data 332 from customer support center database 330 to control the presentation of information to the agent via the client. For instance, the appearance of the toolbar presented by the client is determined according to configuration data 332. The buttons that appear, the commands that are invoked when an agent clicks each button, and the response triggered by an incoming event are all specified as part of configuration data 332 by an administrator using configuration user interface 340.

Fig. 4 shows an example of the operation of components of client/server system 100 to establish a web collaboration session between a customer and an agent. In step 1, a customer requests a live web collaboration session with an agent. Web collaboration driver 120G generates a WebCollabArrived event in step 2, and sends the WebCollabArrive event to session mode communication server 110, as shown in step 3. In step 4, session mode communication receiver 110 receives the WebCollab Arrived Event and, in step 5, determines an appropriate event response. To determine the event response, the originating channel driver for the event is determined as described above by querying customer support center database 330 (query not shown). In this case, the event response is to perform a notification function, as shown in step 6, to provide a notification to the agent via web browser client 104, as shown in step 7. An example of a notification is to cause a button on toolbar 105 to blink and/or to provide a data window with information about the customer and the web collaboration request.

In step 8, the agent accepts the web collaboration request by activating a user interface object such as a work item object of toolbar 105, such as clicking on an accept work item button. The work item object is associated with a command, here an AcceptWebCollab command, that is sent in step 9 to session mode communication server 110. Session mode

communication server 110 sends the AcceptWebCollab command to web collaboration driver 120G as shown in step 10, which performs the AcceptWebCollab command as shown in step 11. In this case, web collaboration driver 120G dynamically establishes web collaboration connection 450 between web server 130I and web browser client 104.

In step 12, web collaboration driver 120G generates a WebCollabStarted event and sends the WebCollabStarted event to session mode communication server 110 in step 13. In step 14, session mode communication server 110 receives the WebCollabStarted event and determines the appropriate event response in step 15. In this case, the event response, as shown in step 16, is to create a record and store it in customer support center database 330. When the web collaboration session is completed, web collaboration driver 120G will generate the appropriate events and send them to session mode communication server 110, which will determine an appropriate event response and perform the event response.

Fig. 5 shows an example of the operation of components of client/server system 100 using the universal queuing system of Fig. 1 to assign an agent to an incoming telephone call and route the telephone call to the assigned agent. In step 1, the customer calls 1-800-company to submit a customer support request. When the call arrives, ACD switch driver 120D detects the incoming telephone call and generates a CallArrived event. While ACD switch 130E is capable of automatically routing the call, in this example inbound communication receiver 170 is configured to allow an automated assignment of agents rather than using the hardware capabilities of the ACD switch driver 130E to route the call.

Inbound communication receiver 170 monitors particular phone numbers including 1-800-company. When inbound communication receiver 170 receives the CallArrived event in step 4, inbound communication receiver 170 determines the originating channel driver 120D as shown in step 5 and determines the event response in step 6. In this case, the event response is to run an e-script, as shown in step 7. In this example, the e-script requests an agent assignment in step 7a, and when the agent assigned message arrives, sends a transfer call command to the originating channel driver 7b. In step 8, the request agent assignment is submitted to UQ system 102 and UQ system 102 assigns an agent in step 9. In step 10, UQ system 102 sends an agent assigned message to inbound communication receiver 170, as described above. Note that several components of system 100 between inbound

communication receiver 170 and UQ system 102 are omitted from the figure, as shown in the breaks in the lines of the arrows between the two components.

Inbound communication receiver 170 receives the agent assigned message in step 11, and, following step 7b of the e-script, sends a transfer call command to ACD switch driver 120D. ACD switch driver 120D performs the TransferCall command and transfers the call to the agent in step 13. In step 14, the agent's phone rings. In step 15, ACD switch driver 120D detects that the agent's telephone handset is ringing and generates a CallRinging event. ACD switch driver 120D sends the CallRinging event to session mode communication server 110 in step 16, which handles notification of the agent of an incoming telephone call.

In step 17, session mode communication server 110 determines an appropriate event response, here to perform a notification function, and in step 18 sends a notification to toolbar 105. In step 19, toolbar 105 notifies the agent of the incoming call, and in step 20, the agent accepts the call by activating an accept work item object. In step 21, an AcceptCall command is sent to session mode communication server 110, which sends the AcceptCall command to ACD switch driver 120D, as shown in step 22. In step 23, ACD switch driver 120D performs the AcceptCall command to connect the customer placing the call with the assigned agent. ACD switch driver 120D will continue to generate events and session mode communication server 110 will continue to perform event responses as long as agents are logged in.

If the agent does not click an accept work item object on toolbar 105, but instead picks up the handset, no AcceptCall command is generated. Instead, ACD switch driver 120D detects that a call has been connected by listening to ACD switch 130E. In such a case, ACD switch driver 120D would generate a CallConnected event and session mode communication server 110 would perform the appropriate event response.

An example of commands implemented by a channel driver 120 for an email / fax server is provided in Table 1 below.

TABLE 1

AcceptEmailFax	For agent to accept the incoming email or fax item. When this device command is invoked, the original media event received at the background-mode Communication Server will be dispatched to Communication Media Manager.

ReleaseEmailFax	For agent to release the active email or fax work item. Then the driver uses SRM to notify UQ server that the agent is ready for the next work item.
TransferEmailFax	For agent to transfer the current email or fax item to another agent. This will be implemented using SRM API.
NotReadyForEmailFax	Set agent to not ready state in UQ system for email or fax. The implementation is the same as "ReleaseEmailFax" using SRM.
AcceptWorkCollab	For agent to accept the incoming web collaboration. When this device command is invoked, the original media event received at the background-mode Communication Server will be dispatched to Communication Media Manager.
ReleaseWorkCollab	For agent to release the incoming web collaboration. Same implementation as "ReleaseEmailFax".
TransferWebCollab	For agent to transfer the current web collaboration session to another agent. (This device command is still open and subject to change)
NotReadyForWebCollab	Set agent to not ready state in UQ system for web collaboration. Same implementation as "NotReadyForEmailFax".

An example of events provided by a channel driver 120 for an email / fax server is provided in Table 2 below.

TABLE 2

EventEmailFaxArrive	Report the arrival of new email or fax
EventEmailFaxConnected	Report the agent has accepted the new email or new fax
EventEmailFaxReleased	Report the agent has released the email of the fax
EventWebCollabArrive	Report the arrival of new web collaboration
EventWebCollabConnected	Report the agent has accepted the new web collaboration
EventWebCollabRelease	Report the agent has released web collaboration
EventAgentReady	Report the agent is ready for a particular media type
EventAgentNotReady	Report the agent is not ready for a particular media type

The multi-channel media independent server as described herein provides many advantages, such as enabling an agent to receiving incoming and send outgoing communication via multiple communication channels of different media types. Customer support requests are received and presented to an agent via a user interface. The agent uses this information to manage active work items, accept work items, release work items, suspend work items, and resume work items.

Other Embodiments

The present invention has been described in the context of software applications running on one or more computer systems. However, those skilled in the art will appreciate that the present invention is capable of being distributed as a program product in a variety of forms, and that the present invention applies equally regardless of the particular type of signal bearing media used to actually carry out the distribution. Examples of signal bearing media include: recordable media such as floppy disks and CD-ROM and transmission media such as digital and analog communication links, as well as media storage and distribution systems developed in the future.

Additionally, the foregoing detailed description has set forth various embodiments of the present invention via the use of block diagrams, flowcharts, and examples. It will be understood by those within the art that each block diagram component, flowchart step, and operation and/or element illustrated by the use of examples can be implemented, individually and/or collectively, by a wide range of hardware, software, firmware, or any combination

thereof. In one embodiment, the present invention may be implemented via Application Specific Integrated Circuits (ASICs). However, those skilled in the art will recognize that the embodiments disclosed herein, in whole or in part, can be equivalently implemented in standard integrated circuits, as a computer program running on a computer, as firmware, or as virtually any combination thereof. Designing the circuitry and/or writing the programming code for the software or firmware would be well within the skill of one of ordinary skill in the art in light of this disclosure.

The present invention is well adapted to attain the advantages mentioned as well as others inherent therein. While the present invention has been depicted, described, and is defined by reference to particular embodiments of the invention, such references do not imply a limitation on the invention, and no such limitation is to be inferred. The invention is capable of considerable modification, alteration, and equivalents in form and function, as will occur to those ordinarily skilled in the pertinent arts. The depicted and described embodiments are exemplary only, and are not exhaustive of the scope of the invention. Consequently, the invention is intended to be limited only by the spirit and scope of the appended claims, giving full cognizance to equivalents in all respects.